Amendments to the Claims:

This listing of claims will replace all prior versions, and listings, of claims in the application:

Listing of Claims:

1. (Currently Amended) A computer system, comprising:

a pipelined, simultaneous and redundantly threaded ("SRT") processor having at least two threads; and

~~an input/output ("I/O") controller coupled to said processor;~~

~~an I/O device coupled to said I/O controller; and~~

a system memory coupled to said processor;

wherein said SRT processor further comprises[[:]]

a load/store execution unit having a store queue~~, the store queue adapted to store~~ that stores memory requests submitted by the at least two threads, where said memory requests change values in system memory directly or indirectly;

a compare logic coupled to said load/store execution unit~~;~~

~~wherein said compare logic~~ that scans the contents of said store queue for corresponding memory requests, and said compare logic verifies that each corresponding memory request matches; and

wherein said compare logic, based on whether the corresponding memory requests match, performs one of ~~1)~~ allowing the memory request to execute~~, and~~ or ~~2)~~ initiating fault recovery.

2. (Original) The computer system as defined in claim 1 wherein said memory requests that directly or indirectly change data values in system memory further comprise at least committed store requests.

3. (Original) The computer system as defined in claim 1 further comprising said SRT processor capable of performing, within each thread independently, program instructions in an order different from the other thread.

4. (Original) The computer system as defined in claim 1 wherein each of said threads of said processor performs speculative branch execution independently from the other.

5. (Currently Amended) A method of checking for transient faults in a simultaneous and redundantly threaded processor having at least two threads, the method comprising verifying, as between the at least two threads, ~~only~~ committed store requests and data load requests from sources that are not cached.

6. (Currently Amended) The method as defined in claim 5 further comprising:
storing a first committed store from a first of the at least two threads;
storing a second committed store from a second of the at least two threads;
comparing at least an address and data field of the first written committed store to at least an address and data field of the second store; and
allowing at least one of the committed stores to execute if the address and data of each of the first and second stores ~~exactly~~ match.

7. (Currently Amended) The method as defined in claim 6 further comprising:
disallowing execution of either of the first or second committed stores if their address and data fields do not ~~exactly~~ match; and
initiating a fault recovery sequence.

8. (Currently Amended) A method of detecting transient faults in a simultaneous~~ly~~ and redundantly threaded microprocessor having at least two threads, the method comprising:

executing a program as a first thread;
generating a first committed store request from said first thread;
storing said first committed store request in a storage queue;
executing the program as a second thread;
generating a second committed store request from said second thread;
storing said second committed store in said storage queue;
checking an address and data associated with said first committed store request against an address and data associated with said second committed store request in a compare logic; and
allowing one of said first and second committed store requests to execute if the checking step shows those committed store requests are ~~exactly~~ the same.

9. (Original) The method as defined in claim 8 wherein executing the first and second threads further comprises executing the first thread a plurality of program steps ahead of the second thread.

10. (Original) The method as defined in claim 9 further comprising:
allowing the first and second threads to make speculative branch execution independent of each other.

11. (Original) The method as defined in claim 9 further comprising:
allowing the first thread to execute program steps out of an order of the program;
allowing the second thread to execute program steps our of the order of the program; and
allowing each of the first and second threads to execute the program in a different order from each other.

12. (Original) A simultaneous and redundantly threaded microprocessor comprising:

a first pipeline executing a first program thread;

a second pipeline executing a second program thread;

a store queue coupled to each of said first and second pipelines;

a compare circuit coupled to said store queue;

wherein each of said first and second program threads independently generate corresponding committed write requests, and each thread places those committed write requests in the store queue; and

wherein said compare circuit detects transient faults in operation of said first and second pipeline by comparing at least the committed store requests from each thread.

13. (Currently Amended) A pipelined, simultaneous and redundantly threaded ("SRT") processor, comprising:

a fetch unit that fetches instructions from a plurality of threads of instructions;

an instruction cache coupled to said fetch unit and storing instructions to be decoded and executed; and

decode logic coupled to said instruction cache to decode the type of instructions stored in said instruction cache;

wherein said processor processes a set of instructions in a leading thread and also in a trailing thread, and wherein the instructions in the trailing thread are substantially identical to the instructions in the leading thread, the instructions in the trailing thread beginning processing through the processor after the corresponding instructions in the leading thread begin processing through the processor;

and wherein said processor detects transient faults by verifying as between the leading and trailing threads ~~only~~ the committed stores and uncached memory read requests.

14. (Currently Amended) A method of detecting transient faults in a simultaneous and redundantly threaded microprocessor having at least two threads, the method comprising:

executing a program as a first thread;

generating a first committed store request from said first thread;

storing said first committed store request in a storage queue;

executing the program as a second thread;

generating a second committed store request from said second thread;

checking an address and data associated with said first committed store request against an address and data associated with said second committed store request; and

allowing one of said first and second committed store requests to execute if the checking ~~step~~ shows those committed store requests are ~~exactly~~ the same.

15. (Original) The method as defined in claim 14 wherein executing the first and second threads further comprises executing the first thread a plurality of program steps ahead of the second thread.

16. (Original) The method as defined in claim 15 further comprising:

allowing the first and second threads to make speculative branch execution independent of each other.

17. (Original) The method as defined in claim 15 further comprising:

allowing the first thread to execute program steps out of an order of the program;

allowing the second thread to execute program steps our of the order of the program; and

allowing each of the first and second threads to execute the program in a different order from each other.

18. (Original) A simultaneous and redundantly threaded microprocessor comprising:

a first pipeline executing a first program thread;

a second pipeline executing a second program thread;

a store queue coupled to at least said first pipelines;

wherein each of said first and second program threads independently generate corresponding committed write requests, at least said first thread places the committed write requests in the store queue; and

wherein said second thread detects transient faults in operation of said first and second pipeline by comparing at least the committed store requests from each thread.